OSSに対する2種類のモデルに基づく最適バージョンアップ問題に関する適合性比較 (不確実な状況における意思決定の理論と応用)

AUTHOR(S):

田村, 慶信; 山田, 茂

CITATION:

田村, 慶信 ...[et al]. OSSに対する2種類のモデルに基づく最適バージョンアップ問題に関する適合性比較 (不確実な状況における意思決定の理論と応用). 数理解析研究所講究録 2008, 1589: 195-204

ISSUE DATE:

2008-04

URL:

http://hdl.handle.net/2433/81569

数理解析研究所講究録
第1589巻 2008年 195-204

OSSに対する2種類のモデルに基づく最適バージョンアップ問題に関する適合性比較

田村 慶信

広島工業大学情報学部
情報工学科
〒731-5193
広島市佐伯区三宅 2-1-1
Phone: 082-921-6042
Fax: 082-921-8978
E-mail: tam@cc.it-hiroshima.ac.jp

山田 茂

鳥取大学工学部
社会開発システム工学科
〒680-8552
鳥取市湖山町南 4-101
Phone: 0857-31-5303
Fax: 0857-31-0882
E-mail: yamada@sse.tottori-u.ac.jp

概要

オープンソースソフトウェアは，社会の情報システムを維持していくにあたり，セキュリティの問題，コストの問題を解決するための有効な選択肢であると考えられ，この数年急激に普及してきている．一方，システム導入後のサポートおよび品質上の問題がオープンソースソフトウェアの普及を妨げる大きな要因として考えられている．特に，オープンソースソフトウェアのリリース候補版において十分な信頼性を確認することは，正式版リリース後のユーザに対する信頼や人気に大きくかかわるだけでなく，オープンソースソフトウェアの保守コストや保守労力の増大に深く関係する．したがって，リリース候補版の信頼性を評価するとともに，最適なバージョンアップ時期を推定することはオープンソースソフトウェア開発において重要な段階となる．本論文では，2種類のソフトウェア信頼度成長モデルに基づいた最適なバージョンアップ時期の推定手法について議論する．

1 はじめに

現在のソフトウェア開発は，クライアント/サーバ・システムやWebプログラミング，オブジェクト指向開発，ネットワーク環境での分散開発といった新しい開発技術が多用されるようになってきている[1]．特に，ネットワーク環境を利用して開発されたオープンソースソフトウェア（Open Source Software，以下OSSと略す）は，世界中の誰もが開発に参加でき，ソースコードが公開され，誰でも自由に改変することが可能なソフトウェアであることから，組込みシステムやサーバ用途として広く採用されている．また，OSSの代表格ともいうべきLinux[1]は市場でサーバ用途として有効であると認められ，この数年急激に普及してきている[2]．特に，OSSは，経済活動および社会活動を行っていく上において，セキュリティおよびコストの観点から非常に有効な選択肢であると考えられている．

オープンソースプロジェクトは，世界中に分散した複数のコンポーネントが，何らかのプラットフォーム上で的確に機能しているとすれば，開発が迅速であるばかりか，競争原理によりある評価基準の下でベストなものが残っていくと考えられる．オープンソースプロジェクトが分散型開発モデルを採用して成功した例として，GNU/Linuxオペレーティングシステムや Apache ウェブサーバなど[2]が挙げられる[3]．

一方，OSSの利用に関しては，未だに多くの不安が残されている．まず第1に，システム導入後のサポートおよび品質上の問題といった利用者側の一般的な不安である．第2に，OSSは本当にビジネスになるのか，オープンソースのソフトウェアを事業化することによって自社製のソフトウェア商品までが市場を失うことにならないか，といった開発者側の不安である[4]．特に，サポートや品質上の問題については，OSSの普及を妨げる大きな要因として考えられている．また，OSSに対する現在の研究動向としては，設計工程や開発手法を対象とした文献はいくつか提案されているが[5, 6]，動的解析に基づいた信頼性評価に関する研究はほとんど行われていないのが現状である．

[1] Linuxは，Linus Torvaldsの米国およびその他の国における登録商標あるいは商標である．
[2] その他記載している会社名，商品名は一般に各社の商標または登録商標である．

本論文では，こうしたオープンソースプロジェクトの下で開発されているOSSに対して，テスト管理に関する問題の1つとして2種類のソフトウェア信頼度成長モデル（software reliability growth model，以下SRGMと略す）[7]に基づいた信頼性評価法について議論する．特に，システム全体を構成する複数のコンポーネント，いわゆる各ソフトウェアコンポーネントの重要度を推定するためにニューラルネットワークを適用する．これにより，バグトラッキングシステム上から得られたデータに基づき，各コンポーネント間の相互作用を包括した信頼性評価法として利用できるものと考える．また，実際のフォールト発見数データに対する数値例を示す．さらに，OSSの最適なバージョンアップ時期の推定方法について考察する．

2 各コンポーネントに対する信頼性評価

ソフトウェアの信頼性評価手法の開発において，各コンポーネントでのデバッグの状況やその良し悪しが，システム全体の信頼性に与える影響を考慮しようとする場合，プログラムパス，コンポーネントの規模，フォールト報告者のスキルなどの，様々に絡み合った要因を捉える必要があると考えられる．こうした複雑な状況下でシステム全体の信頼性に対する各コンポーネントの影響度合いを推定するために，本論文ではニューラルネットワークを適用する．これにより，バグトラッキングシステム上から採取されたデータのみに基づいて機械的に各コンポーネントのシステム全体に与える重要度を推定することが可能となる．

OSSにおいて，システム全体の信頼性に対する各コンポーネントの影響を考えた場合，コンポーネントの規模，フォールト報告者のスキル，フォールト修正の状態，コンポーネントの開発時間，コンポーネント間のパスの数，コンポーネント間の入出力データ量といった様々な要因を考慮する必要がある．こうした複雑な状態を適当な仮定の下で物理的な意味からモデル化することは困難である．したがって，本論文では，各コンポーネント間の内部状態をブラックボックスとして捉えるためにニューラルネットワーク[8]を適用する．すなわち，入力と出力の関係から，その内部構造をニューラルネットワークにより学習させることによって，各コンポーネントがシステム全体の信頼性に与える影響度合いを推定する．これにより，バグトラッキングシステム上から採取されたデータのみに基づいた信頼性評価が可能となることから，実利用上においても容易に適用できるものと考える．本論文においては簡単のために3層ニューラルネットワークを適用する．

まず，$w_{ij}^1(i=1,2,\cdots,I;j=1,2,\cdots,J)$を入力層と中間層の結合係数，また$w_{jk}^2(j=1,2,\cdots,J;k=1,2,\cdots,K)$は中間層と出力層の結合係数とする．さらに，$x_i(i=1,2,\cdots,I)$は正規化された入力データを表し，本論文では，フォールト報告者により致命的であると判断されたフォールト数，特定のOSにおいて発見されたフォールト数，システムの内部構造に習熟した修正者のフォールト修正数，システムの内部構造に習熟した発見者のフォールト発見数とした．ここで，入力層，中間層，出力層におけるユニットの数を，各々I個，J個，およびK個とする．また，各々の層のユニットを示すインデックスをi，j，およびkとする．ここで，各々の層のユニットの出力をh_j, y_kとすると，

$$h_j = f\left(\sum_{i=1}^{I} w_{ij}^1 x_i\right), \tag{1}$$

$$y_k = f\left(\sum_{j=1}^{J} w_{jk}^2 h_j\right), \tag{2}$$

となる．但し，$f(\cdot)$はシグモイド型関数であり，

$$f(x) = \frac{1}{1+e^{-\theta x}}, \tag{3}$$

として表される．ここで，θはゲインと呼ばれる定数である．ネットワークの学習を行うために，誤差逆伝播法を用いる．ニューラルネットワークの出力層における値を$y_k(k=1,2,\cdots,K)$とし，教師パターンを$d_k(k=1,2,\cdots,K)$とすると，式(2)のy_kの評価は次式で与えられる．

$$E = \frac{1}{2}\sum_{k=1}^{K}(y_k - d_k)^2. \tag{4}$$

ここで，教師パターン $d_k(k=1,2,\cdots,K)$ には，各コンポーネントにおける累積発見フォールト数データの正規化された値を採用する．すなわち，各コンポーネントにおける累積フォールト発見数データに基づいて，各コンポーネントの重み係数とそれに影響を及ぼす要因の結合状態の特徴をニューラルネットワークの結合係数に蓄積させ，ある時点における各コンポーネントの重要度の推定・予測が可能なモデルを考える．式 (4) の条件のもとに，各結合係数が最急降下法にて以下のように決定される．

$$w_{jk}^2(\sigma+1) = w_{jk}^2(\sigma)+\epsilon(y_k-d_k)\cdot f'\left(\sum_{j=1}^{J} w_{jk}^2(\sigma)h_j\right)h_j, \tag{5}$$

$$w_{ij}^1(\sigma+1) = w_{ij}^1(\sigma)+\epsilon\sum_{k=1}^{K}(y_k-d_k)\cdot f'\left(\sum_{j=1}^{J} w_{jk}^2(\sigma)h_j\right)\cdot w_{ij}^1(\sigma)f'\left(\sum_{i=1}^{I} w_{ij}^1(\sigma)x_i\right)x_i. \tag{6}$$

ここで，σ は更新のサイクル，ϵ は学習の係数を表す．式 (5) および式 (6) の更新により求められた w_{ij}^1 および w_{jk}^2 から，式 (1) および式 (2) により，ニューラルネットワークの出力層における値 $y_k(k=1,2,\cdots,K)$ を算出することができる．これにより，各コンポーネントに対する重みパラメータ $p_i(i=1,2,\cdots,K)$ を次の式により導出できる．

$$p_i = \frac{y_i}{\sum_{i=1}^{K} y_i}. \tag{7}$$

本論文では，ニューラルネットワークにおいて推定された各コンポーネントの重要度 p_i を，システム全体の信頼性に対する各コンポーネントの影響率を表すものとする．

3 OSS に対する SRGM

3.1 一般化非同次ポアソン過程モデル

従来から，ソフトウェアの信頼性を定量的に評価する手法として，ソフトウェア故障の発生現象を不確定事象として捉えて確率・統計論的に取り扱う方法がとられている．その 1 つが，SRGM である [7]．中でも非同次ポアソン過程（nonhomogeneous Poisson process，以下 NHPP と略す）モデルは，実利用上極めて有効でありモデルの簡潔性が高いゆえにその適用性も高く，実際のソフトウェア信頼性評価に広く応用されている．この NHPP モデルは，所定の時間区間内に発見されるフォールト数や発生するソフトウェア故障数を観測して，これらの個数を数え上げる計数過程 $\{N(t),t\geq 0\}$ を導入し，以下の式で与えられる確率変数すなわちポアソン過程を仮定する SRGM である [7]．

$$\Pr\{N(t)=n\} = \frac{\{H(t)\}^n}{n!}\exp[-H(t)] \qquad (n=0,1,2,\cdots). \tag{8}$$

ここで，$\Pr\{\cdot\}$ は確率を表し，$H(t)$ は時間区間 $(0,t]$ において発見される総期待フォールト数，すなわち $N(t)$ の期待値を表し，NHPP の平均値関数と呼ばれる．

特に，オープンソースプロジェクトについて考えた場合，通常のソフトウェア開発のテスト工程で適用される SRGM のように，ソフトウェア内に潜在するフォールト数が有限であると仮定されたモデルよりも，むしろ無限回のソフトウェア故障が発生すると仮定されたモデルを適用する方が現実的である．すなわち，オープンソースプロジェクトでは常にフォールト修正やバージョンアップが繰り返されており，使用頻度や人気の高い OSS になるほどフォールト報告が頻繁に行われている．この運用形態はオープンソースプロジェクトが無意味なものとみなされ解散するまで続けられる．したがって，1 つの企業組織内において，ある特定の使用目的に限定されたソフトウェアの開発を対象としている従来の SRGM により，OSS の信頼度成長現象を包括することは難しくなる [9, 10]．

本論文では，検出可能フォールト数が無限であると仮定された NHPP に基づく一般化された対数型ポアソン実行時間モデルを適用する [7]．

このモデルでは，時間区間 $(0,t]$ で発見される総期待フォールト数を表す平均値関数 $\mu(t)$ は，

$$\mu(t) = \frac{1}{\theta - P}\ln[\lambda_0(\theta - P)t + 1]$$
$$subject\ to\ \ (P-\theta) < \frac{1}{\lambda_0 t} \qquad (0<\theta,\ 0<\lambda_0,\ 0<P<1), \tag{9}$$

により与えられる．ここで，パラメータ λ_0 は初期故障強度，パラメータ θ はソフトウェア故障1個当りの故障強度の減少率を表す．また，パラメータ P はシステム全体に及ぼすコンポーネントの影響率を表す．上記のNHPPモデルから，種々のソフトウェア信頼性評価のための定量的尺度を導出できる．

3.2 一般化確率微分方程式モデル

本論文では，確率微分方程式から導出されたソフトウェア信頼度成長モデルを提案する．まず，時刻 $t=0$ でOSSがリリースされ，任意の時刻 t におけるソフトウェア内の発見フォールト数 $\{S(t), t \geq 0\}$ は以下の常微分方程式によって記述されるものと仮定する．

$$\frac{dS(t)}{dt} = \lambda(t)S(t). \tag{10}$$

ここで，$\lambda(t)(>0)$ は時刻 t におけるソフトウェア故障強度を表す．

OSSのフォールト報告では，フォールトの発見と同時にバグトラッキングシステムへのフォールト情報の登録が行われるというわけではなく，フォールト発見の前後に若干のタイムラグが生じた状態で登録が行われる場合が多い．このように，バグトラッキングシステム上へのフォールトの登録を考えた場合，OSSのフォールト発見事象は，常に不規則な状態であると考えられる．こうした不規則性を，標準化されたGauss型白色雑音によって近似的に表現することを考える．さらに，OSSは常にバグフィックスやコンポーネントの加除が繰り返されており，ソフトウェア故障強度もそれに応じて変化するものと考えられる．

上記のことから，故障強度 $\lambda(t)$ に不規則性を導入すると，式(10)は，

$$\frac{dS(t)}{dt} = \{\lambda(t) + \sigma\gamma(t)\}\, S(t), \tag{11}$$

となる．ここで，$\sigma\ (>0)$ は定数パラメータであり，$\gamma(t)$ は解過程のMarkov性を保証するために標準化されたGauss型白色雑音である．さらに，$\lambda(t)$ は時刻 t におけるソフトウェア故障強度関数を表す．式(11)を以下のItô型[11, 12]の確率微分方程式に拡張して考える[13].

$$dS(t) = \{\lambda(t) + \frac{1}{2}\sigma^2\}S(t)dt + \sigma S(t)d\omega(t). \tag{12}$$

式(12)の確率微分方程式を初期条件 $S(0)=v$ の下でItôの公式[11, 12]を用いて変換すると，

$$S(t) = v \cdot \exp\left(\int_0^t \lambda(s)ds + \sigma W(t)\right), \tag{13}$$

となる．ここで，v は以前のバージョンまでに発見されたフォールト数を表す．式(13)で定義された $W(t)$ の性質は，次の通りである．

(1) $W(t)$ はGauss過程である．

(2) $W(t)$ の平均および分散は，それぞれ

$$\mathrm{E}[W(t)] = 0, \mathrm{Var}[W(t)] = \sigma^2 t, \tag{14}$$

により与えられる．

(3) $W(t)$ は定常独立増分をもつ．

(4) $\Pr[W(0)=0]=1$.

本論文では，ソフトウェア故障強度関数を以下のように仮定する．

$$\int_0^t \lambda(s)ds = (1 - \exp[-\alpha t]). \tag{15}$$

ここで，α はソフトウェア故障強度の加速係数を表す．式 (15) の強度関数は，その他にも S 字形などの形状を仮定することも可能である．しかしながら，OSS のフォールト報告の特徴として，バージョンアップ直後においては，フォールト報告数は指数関数的に増加するという経験的傾向があることから，これを反映するために指数形の強度関数を仮定した．さらに，モデルの構築および適用可能性に関する考察に重点を置くために，単純な構造をもつ式 (15) の強度関数について取り扱うこととする．

式 (13) から，

$$\lim_{t\to\infty} \mathrm{E}[S(t)] = \infty. \tag{16}$$

となり，時刻 t を ∞ と考えた場合の発見フォールト数は ∞ となることが分かる．

提案された一般化確率微分方程式（stochastic differential equation，以下 SDE と略す）モデルから，任意の時刻 t における発見フォールト数の期待値 $\mathrm{E}[S(t)]$ は，

$$\mathrm{E}[S(t)] = v \cdot \exp\left(\int_0^t \lambda(s)ds + \frac{\sigma^2}{2}t\right), \tag{17}$$

となる．

3.3 ソフトウェア信頼性評価尺度

上述した一般化 NHPP モデルおよび一般化 SDE モデルから，種々のソフトウェア信頼性評価のための定量的尺度を導出できる．

瞬間フォールト発見率は強度関数により表すことができる．これは，単位時間当りに発見されるフォールト数として定義される．瞬間フォールト発見率は，一般化 NHPP モデルから以下のように導出できる．

$$\mu_d(t) = \frac{d\mu(t)}{dt}. \tag{18}$$

また，一般化 SDE モデルから，$S(t)$ の分散を次のように求めることができる．

$$\mathrm{Var}[S(t)] = \mathrm{E}[\{S(t) - \mathrm{E}[S(t)]\}^2] = v^2 \exp\left(2\int_0^t \lambda(s)ds + \sigma^2 t\right)\left\{\exp(\sigma^2 t) - 1\right\}. \tag{19}$$

平均ソフトウェア故障発生時間間隔（mean time between software failures: MTBF）は，ソフトウェア 故障の発生頻度を表すのに有益な尺度である．また，MTBF が大きな値を取ることは，それだけフォールトが発見し難くなり，ソフトウェア信頼性が向上したと判断できることになる．任意の時刻 t における瞬間 MTBF（instantaneous MTBF: $\mathrm{MTBF_I}$）および累積 MTBF（cumulative MTBF: $\mathrm{MTBF_C}$）は，一般化 NHPP モデルおよび一般化 SDE モデルから，以下のように導出できる．

任意の時刻 t における瞬間的なフォールト発見間隔の平均を意味する瞬間 MTBF は，

$$MTBF_I(t) = \frac{1}{d\mu(t)/dt}, \tag{20}$$

$$MTBF_I(t) = \mathrm{E}\left[\frac{1}{dS(t)/dt}\right], \tag{21}$$

となる．

運用開始時点から考えたときの発見フォールト 1 個当りに要する発見時間の平均を意味する累積 MTBF は，

$$MTBF_C(t) = \frac{t}{\mu(t)}, \tag{22}$$

$$MTBF_C(t) = \mathrm{E}\left[\frac{t}{S(t)}\right], \tag{23}$$

により表すことができる．

4 最適バージョンアップ

4.1 最適バージョンアップ時期の推定

OSS の開発において，ある程度目安となるような適切なバージョンアップ時期を推定することは，リリース後の信頼性維持や進捗度管理に役立つと考えられる．本論文では，OSS のバージョンアップ時期の推定方法について議論する．

バージョンアップには大きく分けてマイナーバージョンアップとメジャーバージョンアップがある．しかしながら，どの程度の改訂で区別されるという明確な基準がある訳ではない．マイナーバージョンアップは，旧版にいくつかの細かい機能が追加されたり，性能が若干向上した場合に実施される．ただし，機能の不具合やセキュリティ上の脆弱性を修復するような，クリティカルなフォールトがいくつか発見され緊急に修正を施す必要がある場合に実施される改訂はマイナーバージョンアップではなく，バグフィックスと呼ばれている．一方，メジャーバージョンアップは，OSS 自体の機能が大きく変更されたり，大型の新機能の追加や，性能が劇的に向上した場合に実施される．OSS におけるマイナーバージョンアップは，不定期かつ頻繁に実施されていることから，その推定時期を予測することは困難であり，たとえマイナーバージョンアップ時期が推定できたとしても，その有益性は小さいと考えられる．したがって，本論文では，メジャーバージョンアップを対象とし，前回のバージョンアップ期間に基づいて，次期バージョンアップ時期を予測するための手法について考察する．

4.2 総開発労力の定式化

OSS の開発に伴う総開発労力を定式化し，総開発労力を最小にする時刻を最適バージョンアップ時刻と定義することにより，バージョンアップ後の信頼性維持や進捗度管理に役立つものと考える [14, 15]．まず，総開発労力を定式化するために，以下のパラメータを定義する．

m_0: フォールト修正に伴う修正労力 $(m_0 > 0)$,

m_1: メジャーバージョンアップ後のフォールト修正に伴う保守労力 $(m_1 > m_0)$,

m_2: メジャーバージョンアップ後の単位時間当りの保守労力 $(m_2 > m_0)$.

よって，以下のようなシステム全体における NHPP モデルおよび SDE モデルに対する期待開発労力が得られる．

$$E_1(t) = m_0 \cdot \mu(t), \tag{24}$$

$$E_1(t) = m_0 \cdot \mathrm{E}[S(t)]. \tag{25}$$

一方，メジャーバージョンアップ後の保守労力は以下のように定式化できる．

$$E_2(t) = m_1\{\mu(t_0) - \mu(t)\} + m_2 t, \tag{26}$$

$$E_2(t) = m_1\{\mathrm{E}[S(t_0)] - \mathrm{E}[S(t)]\} + m_2 t. \tag{27}$$

ここで，t_0 は過去のバージョンアップ期間の平均値を表す．

さらに，時間区間 $(0, t_0]$ を越えた場合において，コンポーネントを新規に開発する際には，整合性を確認するためのペナルティ労力が課せられるものとする．本論文では，ペナルティ関数を以下のように定義する．

$$G(t) = m_3 \cdot c \cdot e^{k(t-t_0)}. \tag{28}$$

ここで，$c(>0)$ は，t_0 を越えて開発されたシステム全体に対する新規コンポーネントの割合，$k(>0)$ は，過去のメジャーバージョンアップ回数，m_3 はペナルティ労力に関する定数パラメータを表す．

したがって，総期待開発労力は，

$$E(t) = E_1(t) + E_2(t) + G(t), \tag{29}$$

のように表すことができる．この式 (29) を最小にする時刻 t^* が，OSS の最適バージョンアップ時刻となる．

表 1： FC7 におけるリリース候補版の開発スケジュール.

Date	Event
1 February 2007	Test1 Release
29 February 2007	Test2 Release
27 March 2007	Test3 Release
24 April 2007	Test4 Release
31 May 2007	Fedora 7 General Availability

表 2： FC7 における重みパラメータの推定結果.

Component Name	Weight parameters p_i
Kernel	p_1=0.882
Kernel-xen	p_2=0.069
Kernel-module-thinkpad	p_3=0.012
Kernel-pcmcia-cs	p_4=0.012
Kernel-utils	p_5=0.012
Kenrel-xen-2.6	p_6=0.012

5 実測データに適用した数値例

5.1 信頼性評価結果

上述してきた OSS 信頼性評価のための各 SRGM に基づいた信頼性評価結果の数値例を示す.

ここでは一例として，Fedora プロジェクト[3]で開発が進められている Fedora Core Linux の Kernel を取り上げる [16]. 本論文では，Fedora Core 7 (FC7) のリリース候補版である Test 1 のリリース以降における信頼性評価結果を示す. FC7 におけるリリース候補版の開発スケジュールを表 1 に示す.

ソフトウェアの信頼性を評価するために，所定の時間区間内に発見されるフォールト数またはソフトウェア故障発生数のデータに基づいた手法がとられているという歴史的経緯 [7] から，本論文におけるニューラルネットワークの出力データと教師信号には，各コンポーネントに対する累積フォールト発見数データを適用する. まず，ニューラルネットワークにより推定された各 Kernel コンポーネントに対する推定結果を表 2 に示す. Fedora Core を構成するコンポーネント数は小規模なものを合わせると数百と非常に多いことから，本論文ではシステムの主要コンポーネントとして Kernel を取り上げることとする. この結果から，Kernel コンポーネントの信頼性に関する重要度が最大であり，Kernel-module-thinkpad, Kernel-pcmcia-cs, Kenrel-utils, および Kernel-xen-2.6 のコンポーネントの重要度が最低であることが分かる. これは，Kernel コンポーネントの成熟度が高く，Kernel-module-thinkpad, Kernel-pcmcia-cs, Kenrel-utils, および Kernel-xen-2.6 のコンポーネントの成熟度が低いことを意味する.

さらに，一般化 NHPP モデルおよび一般化 SDE モデルに対する推定された累積フォールト発見数の期待値の推定値を図 1 に示す.

5.2 最適バージョンアップ時刻の推定結果

一般化 NHPP モデルおよび一般化 SDE モデルに含まれる各パラメータの推定値がニューラルネットワークおよび最尤法を用いて推定されたという前提の下で，最適バージョンアップ時期推定の数値例を示す. 評価版において十分な信頼性を確認することは，正式版リリース後のユーザに対する信頼や人気に大きく関係すると同時に，OSS の保守コストや保守労力の増大に関係することから，リリース候補版の信頼性評価は正式版リリースへ向けての重要な段階となる.

最適メジャーバージョンアップ時期推定の数値例として，FC7 を取り上げる. 本論文執筆時点において，Fedora

[3]Fedora は Red Hat Inc. の登録商標である.

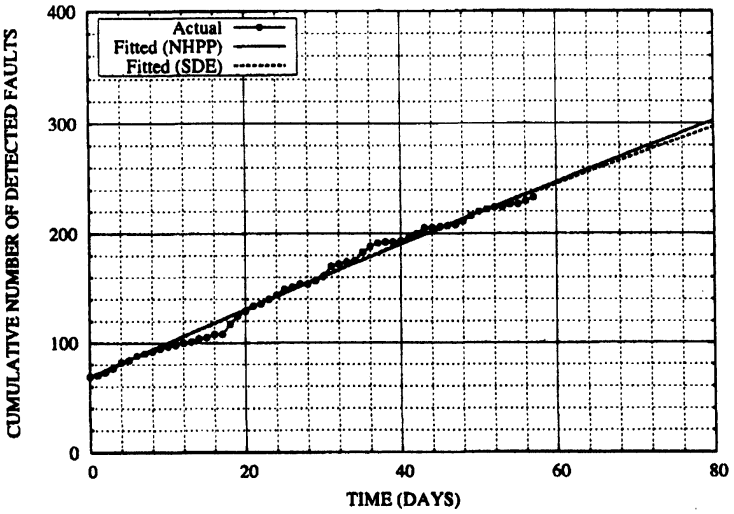

図 1： 一般化 NHPP モデルおよび一般化 SDE モデルにより推定された累積フォールト発見数の期待値.

Core Linux は過去 6 回のバージョンアップを経験している．ここでは一例として，FC6 までの過去のデータに基づき，FC7 への次期バージョンアップ時刻を推定する．2007 年 2 月 1 日以降における各モデルに対する式 (29) の推定された総期待開発労力を図 2 に示す．図 2 から，総期待開発労力を最小にする最適バージョンアップ時刻は FC7 の評価版がリリースされてから，一般化 NHPP モデルの場合は 133 日後となり，そのときの総期待開発労力は 21498 人・日であることが確認できる．一方，一般化 SDE モデルの場合は 139 日後となり，そのときの総期待開発労力は 25019 人・日であることが確認できる．このことから，一般化 SDE モデルに基づいた最適バージョンアップ時期推定手法の方が悲観的な結果が得られている様子が確認できる．

なお，実際の FC7 は，2007 年 2 月 1 日に評価版がリリースされ，120 日後の 5 月 31 日に正式版がリリースされている．このことから，総期待開発労力を最小にするという観点から考えた場合，約 2 週間早い正式版リリースであったことが読み取れる．

6 おわりに

本論文では，オープンソースプロジェクトの下で開発されている OSS に対する信頼性評価法について議論した．特に，ニューラルネットワークを適用することにより，各コンポーネントに対する相互作用を包括した信頼性評価法について議論した．本手法により，システム全体の信頼性に与える各コンポーネントの重要度を定量的に導出することが可能となる．さらに，導出された各コンポーネントの重み係数から各コンポーネントに対する発見フォールト数を推定することが可能となることから，OSS の信頼性を定量的に把握するための手段として有効であると考える．

また，OSS に対する一般化 NHPP モデルおよび一般化 SDE モデルを提案した．さらに，実際の OSS のバグトラッキングシステムから採取されたフォールト発見数データに対する数値例を示した．

OSS の開発において，ある程度目安となるような適切なバージョンアップ時期を推定するために，提案された一般化 NHPP モデルおよび一般化 SDE モデルに基づく最適バージョンアップ時刻推定手法を提案した．本論文において提案された手法によって，OSS のバージョンアップ後の信頼性維持や進捗度管理に役立つと考えられる．特に，最適バージョンアップ時期の推定のために，一般化 NHPP モデルおよび一般化 SDE モデルに基づき OSS の総期待開発労力を定式化した．今後は，フォールト修正に伴う修正労力や単位時間当りの開発労力に関するパラメータの厳密な設定方法について定義する必要がある．

オープンソースという開発スタイルは，今後も何らかの形で市場で大きな流れを作っていくものと考えられる．

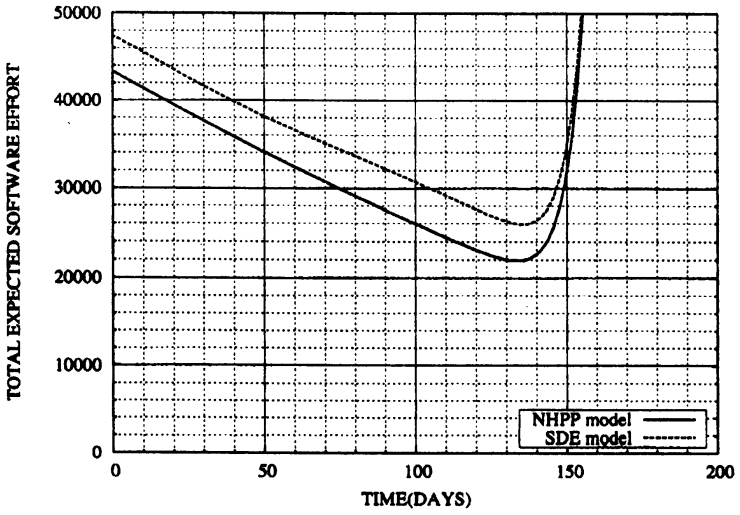

図 2： 一般化 NHPP モデルおよび一般化 SDE モデルにより推定された総期待開発労力.

この流れを阻害する大きな要因として，サポートや品質上の問題が挙げられる．本論文では，こうした問題を解決するために，オープンソースプロジェクトの下で開発された OSS に対する信頼性評価法および最適バージョンアップ時刻推定手法の 1 例を示した．

将来的には，本手法をソフトウェアツールとして実装することにより，多くのユーザに対して容易に扱うことが可能な信頼性評価ツールとして提供していくことが重要であると考える [17]．これまで，OSS では信頼性を定量的に評価するという試みが行われていなかったことから，本論文において新たに提案された信頼性評価手法を OSS に適用することによって，より高品質な OSS の開発に結びつくものと考える．

謝辞

本論文の一部は，文部科学省科学研究費基盤研究 (C)（課題番号 18510124）および若手研究 (B)（課題番号 17700039）の援助を受けたことを付記する．

参考文献

[1] A. Umar, *Distributed Computing and Client-Server Systems*, Prentice Hall, New Jersey, 1993.

[2] トロン協会 ITRON 仕様検討グループ, ITRON Project Archive, http://www.sakamura-lab.org/TRON/ITRON/home-j.html

[3] E-Soft Inc., Internet Research Reports, http://www.securityspace.com/s_survey/data/index.html

[4] 情報技術標準化研究センター, オープンソースソフトウェアの標準化調査研究 成果報告書, 財団法人 日本規格協会, 2007.

[5] A. MacCormack, J. Rusnak, and C.Y. Baldwin, "Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code," *INFORMS Journal of Management Science*, vol. 52, no. 7, pp. 1015–1030, 2006.

[6] Y. Zhoum and J. Davis, "Open Source Software Reliability Model: An Empirical Approach," *Proceedings of the Workshop on Open Source Software Engineering (WOSSE)*, vol. 30, no. 4, 2005, pp. 67–72.

[7] 山田 茂, ソフトウェア信頼性モデル－基礎と応用－, 日科技連出版社, 東京, 1994.

[8] E. D. Karnin, "A Simple Procedure for Pruning Back-propagation Trained Neural Networks," *IEEE Trans. Neural Networks.*, vol. 1, pp. 239–242, 1990.

[9] 田村慶信, 山田茂, 木村光宏, "オープンソース共同開発環境に対するソフトウェア信頼性評価法に関する考察," 電子情報通信学会論文誌, vol. J88–A, no. 7, pp. 840–847, 2005.

[10] Y. Tamura and S. Yamada, "Comparison of Software Reliability Assessment Methods for Open Source Software," *Proceedings of the 11th IEEE International Conference on Parallel and Distributed Systems (ICPADS2005)–Volume II*, Fukuoka, Japan, July 20–22, 2005, pp. 488–492.

[11] L. Arnold, *Stochastic Differential Equations–Theory and Applications*, New York, John Wiley & Sons, 1974.

[12] E. Wong, *Stochastic Processes in Information and Systems*, New York, McGraw–Hill, 1971.

[13] S. Yamada, M. Kimura, H. Tanaka, and S. Osaki, "Software Reliability Measurement and Assessment with Stochastic Differential Equations," *IEICE Trans. Fundamentals*, vol. E77-A, no. 1, pp. 109-116, 1994.

[14] S. Yamada, H. Narihisa, and S. Osaki, "Optimal Release Policies for a Software System with a Scheduled Software Delivery Time," *International Journal of Systems Science*, vol. 15, no. 8, pp. 905–914, 1984.

[15] S. Yamada and S. Osaki, "Optimal Software Release Policies with Simultaneous Cost and Reliability Requirements," *European Journal of Operational Research*, vol. 31, no. 1, pp. 46–51, 1987.

[16] Red Hat, Inc. and others, "Fedora Project", [Online]. Available: http://fedora.redhat.com/.

[17] 田村慶信, 肌附康司, 山田茂, 木村光宏, 「オープンソースソフトウェアに対する最適バージョンアップ時期推定のためのソフトウェアツール」, Japan Linux Conference 2007, 抄録集第 5 巻, 2007 年 9 月 13 日–9 月 14 日, pp. 10–19.